DOCUMENT RESUME

ED 388 260 IR 017 409

AUTHOR Jones, Mark K.; And Others
TITLE A Re-Usable Algorithm for Teaching Procedural Skills.
PUB DATE 94
NOTE 7p.; In: Educational Multimedia and Hypermedia, 1994. Proceedings of ED-MEDIA 94--World Conference on Educational Multimedia and Hypermedia (Vancouver, British Columbia, Canada, June 25-30, 1994); see IR 017 359.
PUB TYPE Reports - Descriptive (141) -- Speeches/Conference Papers (150)

EDRS PRICE MF01/PC01 Plus Postage.
DESCRIPTORS *Algorithms; *Authoring Aids (Programming); *Computer Assisted Instruction; Computer Software Development; Instructional Design; Simulated Environment; Situational Tests; *Teaching Skills
IDENTIFIERS Prototypes

ABSTRACT

The design of a re-usable instructional algorithm for computer based instruction (CBI) is described. The prototype is implemented on IBM PC compatibles running the Windows(TM) graphical environment, using the prototyping tool ToolBook(TM). The algorithm is designed to reduce development and life cycle costs for CBI by providing an authoring environment suited for subject matter experts who do not have instructional skills, and by supporting rapid prototyping. The strategy and tactics are predefined; the instructional developer need only describe the desired performance and the environment of the performance. The specific algorithm described implements a simulation-based reactive environment for learning and practicing device operation skills. Examples would include the operation of many electronic or mechanical devices. The reaction approximates the effects the learner's action would cause in the real environment. The realistic reaction may be augmented by explicit instructional guidance and feedback that would not occur outside the instructional setting. The algorithm begins with a media presentation, then continues with a short tutorial designed to both introduce the procedure and to acquaint the student with some of the learner control capabilities. After the tutorial, the heart of the instruction commences, organized around a set of performances: a demonstration, three levels of practice, and a self-check. Analysis and resource preparation are discussed as two authoring activities. The approach to designing and implementing the algorithm is general, and should apply to other instructional outcomes. (Contains 13 references.) (Author/MAS)

****
* Reproductions supplied by EDRS are the best that can be made *
* from the original document. *
****

ED 388 260

A Re-usable Algorithm for Teaching Procedural Skills

MARK K. JONES, ANDREW S. GIBBONS
Department of Instructional Technology
Utah State University, Logan Utah 84322-2830

DENISE C. VARNER
Southwest Research Institute
6220 Culebra Road, San Antonio , TX 78228-0510

U.S. DEPARTMENT OF EDUCATION
Office of Educational Research and Improvement
EDUCATIONAL RESOURCES INFORMATION CENTER (ERIC)

☐ This document has been reproduced as received from the person or organization originating it.

☐ Minor changes have been made to improve reproduction quality.

• Points of view or opinions stated in this document do not necessarily represent official OERI position or policy.

Abstract: The design of a re-usable instructional algorithm for computer-based instruction (CBI) is described. The algorithm is designed to reduce development and life cycle costs for CBI by providing an authoring environment suited for subject matter experts who do not have instructional skills, and by supporting rapid prototyping. The specific algorithm described implements a simulation-based reactive environment for learning and practicing device operation skills. The approach to designing and implementing the algorithm is general and should apply to other instructional outcomes.

Problem Statement

The opportunity for repeated practice is critical to the development of procedural skills. The effectiveness of practice may be increased by providing guidance and feedback to the learner and by structuring the practice environment to transition in small steps from simple to complex practice situations.

Utilizing a computer to provide practice environments has several advantages. In comparison with the real environment, a computer may be more convenient, safer, and less expensive. In comparison with human-based instructional methods, a computer is more patient, more consistent, and available at a time and location convenient for the individual learner. In comparison to other media-based instructional methods, a computer may provide a reactive practice environment and the potential to customize the instruction to the individual learner.

A reactive practice environment is key to learning procedural skills. The learner performs and the environment reacts. The reaction approximates the effect the learner's action would cause in the real environment. This provides context for the learner and prepares for the transition from the practice to the real environment. The realistic reaction may be augmented by explicit instructional guidance and feedback that would not occur outside the instructional setting. Both learner and the instructional control program may vary the conditions of the environment and the requirements for performance based upon the learner's progress.

Unfortunately, developing computer-based instruction which incorporates a reactive environment, explicit guidance and feedback, and variable practice conditions and performance requirements, is very expensive. Highly specialized personnel are required. The high cost and long timelines of development and the scarcity of skilled personnel limit the application of this type of instruction. It is simply not cost-effective if the potential audience is small, the instruction will have a short useful life, or the development is being undertaken by a small company or intended for public education.

The goal of our research is to dramatically lower the cost of development and the required skill level of development personnel. The approach is to provide re-usable instructional programs in which the strategy and tactics are pre-defined for a particular instructional objective. The instructional developer need only describe the desired performance and the environment of the performance.

PERMISSION TO REPRODUCE THIS MATERIAL HAS BEEN GRANTED BY

Gary H. Marks

TO THE EDUCATIONAL RESOURCES INFORMATION CENTER (ERIC).

IR017409

BEST COPY AVAILABLE

Approach

Existing computer-based instruction (CBI) authoring tools successfully address a specific problem. It is assumed that the author, someone with teaching or instructional design experience, knows what they want to teach and how they want to teach it -- in fact they may already be teaching the material using non-computer means. The author is not however an experienced computer programmer, and does not know how to program the computer to deliver the desired instruction. The authoring language or system is designed to help the author make the translation from what to teach and how to teach, which is known, to how to make the computer teach, which is unknown, and to accomplish this without requiring the author to become an expert programmer.

As computers have become more widely available, more organizations have desired to develop computer-based instruction than there are available teachers and instructional designers. It is now commonly the case that the person responsible for developing CBI is not only not a programmer, but is not an experienced teacher or instructional designer either. The skill this person is most likely to have is subject matter expertise. The assumptions made by existing authoring systems -- that the author knows what to teach and how to teach it -- are no longer valid. In fact, lack of knowledge in these two areas may be the most important barrier to the development of effective CBI.

The successful solution to the problem of teachers who did not know how to program computers was not to teach them to be programmers, but rather to provide tools that allowed them to create CBI without needing to learn much about programming. Similarly, the solution to the problem of subject matter experts who do not know much about instruction will not be to teach them to be teachers, but to provide tools which allow them to create CBI without needing to learn either programming or instructional design.

This approach does not focus on making tools that are just "simple to use". Teaching and learning are complex human endeavors with many variables and incomplete knowledge. This complexity cannot be entirely avoided, however the nature of the complexities presented to authors can be managed. The most important step that can be taken is to assure that the authoring decisions, whether simple or complex, require only knowledge that it is reasonable for the author to have. In other words, the context of the authoring should be familiar to the author. In the case of a subject matter expert, it is feasible to require authors to describe the desired performance and the objects upon which that performance will occur. While some authoring tasks may require authors to think more deeply about these and to describe them more explicitly than they have before, the domain is at least familiar and accessible. On the other hand, even fairly simple authoring tasks that require instructional or programming expertise are inaccessible to the average subject matter expert.

Instruction

We have developed a prototype re-usable instructional algorithm. The prototype is implemented on IBM PC compatibles running the Windows™ graphical environment, using the prototyping tool ToolBook™.

The algorithm is designed to instruct algorithmic procedures in the area of device operation. Examples would include the operation of many electronic or mechanical devices. The specific test case for the prototype was the performance of the Central Air Data Computer Self-Test procedure for F-16 aircraft.

The algorithm begins with a media presentation. This, and all similar media presentations, is completely under the control of the author and may range from a piece of text or a graphic to a complete multi-media presentation including digital audio and video.

The lesson then continues with a short tutorial. This is designed both to introduce the procedure and to acquaint the student with some of the learner control capabilities. The learner is shown general information about the procedure and the sequence of steps that make up the procedure. The results dialog box, which will display the learner's progress, is introduced. The learner is also shown a list of all personnel involved in the procedure and (under author control) may select which role to practice. Each set of information is presented in a dialog box that is keyed to one of the icons in the toolbar. After this introductory tutorial, only the results dialog will be automatically displayed again, however all dialogs are available at the learner's discretion throughout the instruction.

After the tutorial, the heart of the instruction commences. This is organized around a set of performances. The author may add, subtract, and customize the performances, but by default there are five: a demonstration, three levels of practice, and a self-check.

Each performance instructs all steps of the procedure. The algorithm is designed to instruct the performance of each step, the sequence of steps, who performs each step, and any tools that are required for the performance of a step. In addition, the algorithm connects the strictly procedural knowledge to process knowledge: information about what is happening to the objects as they are being acted upon in each step.

The demonstration performance introduces the steps and their sequences. All procedural information is presented to the learner: the name of the step, its order in the sequence, the identity of the person responsible for performing the step, any tools used in performing the step, and a mediated demonstration of the step being performed. The only performance required of the learner is to repeat the performance of the step immediately following the demonstration. Process knowledge is not instructed.

The practice levels slowly fade the cues and increase the requirements for the performance. Additionally, process knowledge is taught. The learner may be asked to predict the value which a property of one of the objects in the environment will have after the step is performed. The learner may also be shown a summary of the effects of a step on all properties simulated by the instruction. From this summary, the learner may request an explanation of the value of any property, or explicit instruction linking sets of input values to output values for a selected property.

Self-check withdraws all cues and requires full performance of the procedure by the learner, including proper ordering of steps, selection of tools, and performance of the steps.

Each performance of the procedure may include an introductory and conclusion media presentation. In addition, at all times that an explicit learner response is not prompted, the learner may explore any object that is visualized or referenced on the screen. This exploration may lead to the display of information, a media presentation, or explicit instruction. For example, clicking on a device brings up a dialog which describes the device and lists its properties and their states.

Step performance is via direct manipulation of a visualization of the performance environment. Each object in the environment (for example, devices, device controls, device indicators) is represented visually by a graphic. Objects which always appear the same are represented by a single graphic. Objects whose appearance varies with their state are represented by a set of graphics, to each of which the author attaches a rule indicating for which state values that graphic is valid. The appropriate graphic is automatically selected for display during instruction based upon the state of the represented object. Objects which represent controls (buttons, switches, knobs, dials, etc.) respond during a learner performance with a popup menu of valid settings when clicked by the learner. The learner then selects a setting. During a demonstration performance, the algorithm selects the setting automatically.

Whenever any action occurs that changes the state of a control, the simulation which underlies the visualization is updated. Each object may have defined upon it one or more properties which represent the state values of the object. For each property the author defines a set of rules which determine the value of that property. During a simulation cycle the rules of all properties are interpreted and the property values updated. Immediately afterward, the rules for all visualizations are re-examined based on the new property values, and the visualizations updated. From the learner's point of view, he or she clicks on a control and selects a setting; the control then changes to represent the new setting (for example, a toggle switch goes from off to on) and possibly other devices change as well (for example, a light may be illuminated).

The simulation is not free play -- the learner may perform any action but only correct actions are simulated. Incorrect actions receive feedback, which may range from a message indicating that the action is not correct to a demonstration of the correct action. The type of feedback is determined based on the performance level and the learner's prior actions. The decision to not support free-play simulation is based primarily on the goal of authoring efficiency. It turns out that the definition of the rules which govern the simulation is the most difficult and least familiar task authors must perform. A free-play simulation requires a rule set considerably more complete and complex than is the case for path-based simulation. A functional and instructionally useful path-based simulation may be constructed with a small set of rules and may be defined with a relatively limited knowledge of the actual inner workings of the device being simulated.

It should be mentioned that an instructional simulation of the type described here is built for different purposes and under different constraints than other simulations. Many simulations, for example those used in engineering design or prediction of natural phenomena, are used to study the system being simulated for the purpose of acquiring new knowledge. These systems are only useful to the extent that they fully and accurately capture the dynamics of the system. An instructional simulation of the type described here is designed to support a reactive environment for practicing the performance of a procedure. It need only account for

existing knowledge and not for new or unknown system attributes. The conditions of its use are constrained. It need not completely model the system, only that part of the system which is instructionally relevant to the objective at hand. Nor must the model be fully accurate, in fact, it may be desirable to implement a model of system function that is simplified as that may be more appropriate for a particular category of learner. For example, the appropriate model for an operator of a device is not the same as the more detailed model necessary for repair or design of the device.

Authoring

Authoring involves two activities: analysis and resource preparation.

Analysis

The analysis required is more involved than is the case with existing authoring systems. The author is required to carefully describe both the performance and the environment of the performance. A set of pre-defined generalized object descriptors, or classes, is provided. These are of three types: content, instructional, and visualization objects. Content objects represent subject matter: examples are procedures, devices, controls, and personnel. Instructional objects define how the instruction is to be carried out; examples include lessons, performances, and guidance settings. A set of pre-defined instructional objects is provided so that authors need not be concerned with these, however for those authors with instructional design expertise the instructional objects are available for customization. Visualization objects link to the resources (graphics, video, audio) which represent the objects of the instructional environment.

Each class defines a set of slots, or attributes, that are valid for objects of that class. Authoring consists of creating new objects (instances) based upon the patterns stored in the classes and providing values for the attributes. Attribute values may be primitive types such as strings or numbers, complex types such as rules, or links to other objects. For example, the step object has attributes for the action to be performed (a complex type which uses a simple grammar to specify a setting for a control, for example "SET main power switch TO on"), links to the person who performs the step and the tools used in the step (links to other content objects), and a link to a visualization object demonstrating the step being performed.

Though the analysis required exceeds that normally performed, there are two important advantages. First, it makes possible the entire approach of re-usable algorithms. The algorithm is defined based upon the object classes, and not upon any specific subject matter. Thus the algorithm is designed to teach a step, its personnel, its tool, its demonstration and action without knowledge of any specific step, personnel, tool, etc. During instruction, the specific elements are retrieved from the knowledge base and substituted. The algorithm may be re-used many times, with different subject matter.

Second, the instruction is generated directly from the analysis. This differs from the standard approach of a sequence of translations, from analysis to design to final development in some programming language. This has several important consequences. With a single representation of the content there are no problems of inconsistent representations: the analysis is always up to date and reflects exactly what is taught. Changes at the analysis level immediately update the instruction. As a result, life-cycle costs are dramatically reduced. Updates and modifications to existing lessons may be carried out at the analysis level, which is the most accessible for humans (as compared to the computer code). Within the knowledge base, each element is represented exactly once, no matter how many times it is used in the instruction. Thus updates to a knowledge object may be made with full assurance that all uses of that object have been updated.

Another important consequence is support for rapid prototyping. Not all attributes need be assigned values in order for instruction to run. In fact, the algorithm is designed to function completely with only a few attributes defined, and even finished instruction may leave many attributes undefined. Missing data is replaced by defaults and place holders. This has the most important effects in the instructional and visualization objects. All instructional objects have pre-defined default values, so that no authoring need be done at all. Visualization objects will provide place holders, so that the instruction may be run with full user interaction prior to the development of all or any graphic or media resources.

Rapid prototyping is an important element of the approach towards reducing overall development costs. Developing prototype instruction very early in the analysis phase means that alternative approaches may be explored with feedback from colleagues and other interested parties. Because the prototype is the instruction,

rather than an abstract representation of the potential instruction (for example, a list of objectives or a storyboard) it is more easily interpreted by those with less expertise in the representations used by instructional developers: for example, management, clients and potential users. Changes suggested by these people may be incorporated at far lower cost than is the case when the first working version of the instruction is ready only when the development schedule is 90% complete.

Resource Preparation

Resource preparation is the development of graphics, video, audio, and any other media to be included in the instruction. The basic work of resource preparation is unchanged: graphic artists must still draw pictures, video production staff must still shoot video.

While the re-usable algorithm approach cannot automate picture drawing, it can positively affect the overall cost of resource development.

First, unlike much current video-based instruction, the resources do not carry the principal instructional message. The algorithm accomplishes this. Resources primarily visualize the environment and supplement the principal message. Fewer overall resources may be needed.

Resources may be re-used. Often, a scene is built from a series of graphics, each of which may be re-used in a variety of situations. Life-cycle costs are reduced when a part of a device changes, as only that part need be re-shot rather than the entire scene in which that part appears.

The rapid prototyping, which may precede resource development, or may use cheap resources such as camcorder video or audio captured by the developer, means that the instruction may be more completely developed and critiqued before resource development commences. This should reduce the need to re-do media production due to design changes.

Finally, the management of the overall development process will have more information available that can be used to streamline media production. Reports may be generated from the knowledge base indicating the number and types of media resources specified in the analysis. This information may be used to coordinate and schedule media production.

Prior Work

This research extends our earlier work on transaction shells (Li & Merrill, 1990; Merrill, Li, & Jones, 1991). This prior work focused on teaching part structure and did not incorporate simulation. The knowledge representation builds on that previously reported in (Jones, Li & Merrill, 1990), which in turn was influenced by work in semantic data bases (Hull & King, 1987; Peckham & Maryanski, 1988) and object oriented programming (Agha, 1987; Goldberg & Robson, 1983). The simulation aspect is influenced by (Towne & Munro, 1988) and (Halff, 1990). The instructional strategy has roots in (Merrill, 1983) and (Gagne, 1985). Notions about mental models are widely published, see for example (White & Frederiksen, 1990). Work on rapid prototyping in software development has some parallels to courseware development, see (IEEE, 1989).

Further Work

This project has completed the first of three phases. In phase 2, a number of extensions and modifications are planned. The most important of these are:

- extensions to the content objects to represent conditional and repeated steps, branches, waits
- instruction of the materials used in a step
- specialized instruction for steps flagged as critical or dangerous
- specialized feedback for actions flagged as common or dangerous errors
- support for all MCI devices
- support for procedural attachments to resources so that they may update themselves
- incorporating structural instruction: device parts and connectivity
- support for multiple environment locations and explicit instruction for finding a control in a location
- maintenance of individual student profiles and customization based on the profile across sessions

We envision eventually creating a library of such re-usable algorithms, of which device operation procedures are just one example.

References

Agha, G. A. (1987). *ACTORS: a model of concurrent computation in distributed systems.* Cambridge, MA: MIT Press.

Gagne, R. M. (1985). *The conditions of learning.* New York: CBS College Publishing.

Goldberg, A. & Robson, D. (1983). *Smalltalk-80: the language and its implementation.* Reading, MA: Addison-Wesley.

Halff, H. (1990). *Automating maintenance training.* Arlington, VA: Halff Resources.

Hull, R., & King, R. (1987). Semantic database modeling: survey, applications, and research issues. *ACM Computing Surveys* 19 (3), 201-60.

IEEE (1989). Rapid prototyping in software development. Special issue of *Computer* 22 (5).

Jones, M. K., Li, Z., & Merrill, M. D. (1990). Domain knowledge representation for instructional analysis. *Educational Technology* 30 (10), 7-32.

Li, Z. & Merrill, M. D. (1990). Transaction shells: a new approach to courseware authoring. *Journal of Research on Computing in Education*, 23 (1), 72-86.

Merrill, M. D., Li, Z., & Jones, M. K. (1991). Instructional transaction theory. *Educational Technology* 31 (6), 7-12.

Merrill, M. D. (1983). Component display theory. In C. M. Riegeluth (Ed.), *Instructional Design Theories and Models.* Hillsdale, NJ: Lawrence Erlbaum.

Peckham, J. & Maryanski, F. (1988). Semantic data models. *ACM Computing Surveys* 20 (3), 153-89.

Towne, D. M. & Munro, A. (1988). The intelligent maintenance training system. In J. Psotka, D. Massey, & S.A. Mutter (Eds.), *Intelligent Tutoring Systems: Lessons Learned.* Hillsdale, NJ: Lawrence Erlbaum.

White, B.Y. & Frederiksen, J.R. (1990). Causal model progressions as a foundation for intelligent learning environments. *Artificial Intelligence* 42, 99-157.

Acknowledgments

This is a team effort and would not be possible without the efforts of a number of people, including M. David Merrill, Zhongmin Li, Scott Schwab, Mark Lacy, Leston Drake, Vicki Napper, Jean Pratt, Richard Cline, and Thor Anderson.

We especially appreciate the support of our sponsors, the Training Systems Program Office of the U.S. Air Force Aeronautical Systems Center.